Short Paper:
A Scalable Complex Event Processing System and Evaluations of its Performance

NEC Corporation
Background

- Enormous number of sensing data can be collected.
- Context-aware services is spreading.
- Many kinds of application wish to utilize useful sensing information.

Platform which can match sensing data and services is required

- real-time processing
- complex event processing
- scalability relative to the number of events (=sensing data)
- scalability relative to the number of rules (=subscription from applications)
Load balancer with local state management

Conventional approach
- Data is distributed by load balancers

Problem
- RR distribution
  - Does not work because CEP is stateful processing
- Hash based distribution
  - Does not work if event_A and event_B don’t have a common key value which is applied to hash function

Complex event can’t be detected, because event_A and event_B are retained in separated state
Load balancer with shared state management

Approach
- Manages state information in shared state
- CEP rule is initiated even if event_A and event_B are sent to separated EPs

Problem
- Access to shared state become bottleneck
Our solution (rule-based event dispatch)

Complex event can be detected.

Dispatches events based on dispatch rules (=Simple event processing rules)

CEP rules are allocated so that load of the servers become even.

Activates CEP rules to servers

Event_A
Partial match

Event_B
Multicast →Waste of resources →Rules should be allocated so that multicasts reduce.

We propose a rule allocation algorithm which solves conflicting condition.

Rule(Event_B & Event_C)
Rule(Event_A & Event_B)

Allocates CEP rules to servers

Server#1
State: R

Server#2
State: Rule(Event_B & Event_C)
Scalable Context Delivery Platform (SCTXPF)

Event Processor Controller (EP-CTL)
- Receives CEP rules from applications
- Distributes the CEP rules to EPs for system scalability
- Sets DISP rule to DISPs according to the CEP rule allocation

Layer 1: Stateless process Dispatcher (DISP)
- Every event is routed to EP(s) based on attribute values of itself

Layer 2: Stateful process Event Processor (EP)
- CEP rules are applied
Rule allocation in EP-CTL

Problems

- If many events dispatched to a certain EP
  - the EP could be a bottleneck
- If CEP rules which deal with the same event are allocated to separated EPs,
  - event copy and multicast
  - waste of the system resources

Approach

- To minimize the number of EPs which require the same events
- To keep the difference between EPs’ process loads under a certain threshold
Rule Distribution Algorithm

Decides EP to set the rule in 3 steps

New Rule: Rule( A & B within t, action )

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of rules</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Req’ed events</td>
<td>A,B,C</td>
<td>A</td>
<td>A,B</td>
<td>A,B,C, D,E,F</td>
</tr>
</tbody>
</table>

- **Step 1:**
  Remove EPs violating max deviation (here, Max deviation: 3)

- **Step 2:**
  Select EPs having most common necessary events

- **Step 3:**
  Select EPs having least necessary event types

- EP1: A, B, C
- EP2: A
- EP3: A, B
- EP4: A, B, C, D, E, F

- EP1: OK
- EP2: OK
- EP3: OK
- EP4: N/A

Empowered by Innovation

Page 8
Summary of the Evaluations

Evaluation of the Compared System

- Compared the proposed rule allocation and RR rule allocation
- Prepared CEP rules which simulate location aware service (car-navi model)
  - The number of rules is 1000
- The result revealed that the proposed rule allocation reduces multicast and improves throughput about 30%

Evaluation with Many-core Servers

- Prepared many-core servers and measured throughput increasing the number of EPs (up to 48 EPs)
  - The number of rules is 10000
- The result showed throughput increases linearly in proportion to the number of EPs and reached 2.7M events/sec
Evaluation of the Compared System

System
- CPU: Core i7 (3.4GHz)
- MEM: 4GB
- OS: Ubuntu 10.04.3

Comparison of system
- Proposed: CEP rules are allocated by the proposed algorithm
- Round Robin (RR): CEP rules are allocated in RR fashion
- Shared state: all EPs share a common state

Rule model
- Random rule model: CEP rules are randomly generated
- Car-navi model: emulates car navigation service
Car-navi model

- **CEP Rules**
  - If event(attr=0000) seq event(attr=0101) within 10 sec, Then notify app
  - If event(attr=0001) seq event(attr=0101) within 10 sec, Then notify app
  - If event(attr=0100) seq event(attr=0101) within 10 sec, Then notify app
  - ...
  - If event(attr=0303) seq event(attr=0202) within 10 sec, Then notify app

- **Map Size:** 100×100

- **# of Shops:** 83

- **# of Rules:** 83×12=996
The result: throughput

**Random rule model**

- Multicast = 3.3
- Multicast = 5.9

**Car-navi model**

- Multicast = 1.0
- Multicast = 1.6

Throughput (random rule model, # of rule = 1000)

Throughput (ITS rule model, # of rule = 996)
The result: CPU usage

CPU usage (Car-navi model)

- DISP
- EP

# of EPs: 1, 2, 3, 4, 5, 6

CPU usage (%): 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
Evaluation with Many-core Servers

**Event Generators**
- 6 servers
- Total 48 core

**DISPs**
- 4 servers
- Total 96 core

**EPs (+pseudo app)**
- 6 servers
- Total 48 core

![Diagram of server connections and specifications](image-url)
The Result: throughput

2,700,000 events/sec
Summary

Large scale context aware services: a new CEP applications

The Scalable Context Delivery Platform (SCTXPF)
- Two tier architecture, ie DISPs (state-less) and EPs (state-ful)
- EP-CTL allocates CEP/DISP rules to EPs/DISPs

The rule allocation algorithm
- Allocates CEP rules to EPs so that state are confined in EPs and the load of EPs are even

The evaluation
- Revealed the SCTXPF is scalable and the rule allocation algorithm improves its scalability
- Achieved 2.7 million event/sec
  - There are rooms for improve the performance
Empowered by Innovation

NEC
What is CEP (Complex Event Processing)?

High level events such as human activity can be inferred from plural low level events such as sensor data.

- e.g. plural events: “church bells ringing”, “a man in a tuxedo”, “a woman in a flowing white gown”, “rice flying through the air”
  - Individual event doesn't make any more sense.
  - If all events happen at the same time, the system can infer a complex event: “a wedding”.

Why CEP?

- Conventional
  - Method: All events are once stored in DB and searched to process.
  - Problem: consume much time to store and search data.

- CEP
  - Method: Detect event pattern on the fly.
  - Advantage: Real-time processing.
  - Enables real-time service
Event Processors and Dispatchers

Event Processors (EPs): CEP
- CEP is executed when an event arrives.
- If the event is partly-matched in some rules, the state is held in the EP.
- If the event is matched in some rules, the action in the rules is executed.

Dispatchers (DISPs): Dispatch events to the EPs whose rules need the events
- DISP rules, i.e., simple event rules, are set by EP-CTL.
- Simple event processing is executed when an event arrives.
- If the event is matched in some rules, the event dispatches to the EPs which the rules show.
  - Unicast: if the event is matched an DISP rule which specifies only one EP
  - Multicast: if the matching rule specifies more than one EPs
  - Filtered out: if there is no matching rule
# Table of Contents

- Background
- Challenging Issues on distributing CEP
- The Proposed Scalable Context Delivery Platform
- Evaluations
- Summary
Table of Contents

- Background
- Challenging Issues on distributing CEP
- The Proposed Scalable Context Delivery Platform
- Evaluations
- Summary
Table of Contents

- Background
- Challenging Issues on distributing CEP
- The Proposed Scalable Context Delivery Platform
- Evaluations
- Summary
Table of Contents

- Background
- Challenging Issues on distributing CEP
- The Proposed Scalable Context Delivery Platform
- Evaluations
- Summary
Example Context-aware Services

These services require 1M events/sec processing

<table>
<thead>
<tr>
<th>Services</th>
<th>Event Volume</th>
<th>Event Volume</th>
<th>Response time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cellular phone location service</strong></td>
<td>830,000 events/sec</td>
<td>50Mpeople(half of cell phone users)÷60sec(1notify/min)=830k event/s</td>
<td>1 sec</td>
</tr>
<tr>
<td><strong>Probe car</strong></td>
<td>1,330,000 events/sec</td>
<td>80Mcars÷60sec (1notify/min)=1330k event/s</td>
<td>10 sec</td>
</tr>
<tr>
<td><strong>Home appliances, Eco monitoring</strong></td>
<td>130,000 events/sec</td>
<td>400k houses×20 appliances(/houses)+60sec (1notify/min)=130k event/s</td>
<td>1 sec</td>
</tr>
<tr>
<td><strong>Agriculture</strong></td>
<td>200,000 events/sec</td>
<td>3M farmers×40%×10 (sensors/farmer)+60 (1notify/min)=200k event/s</td>
<td>1 min</td>
</tr>
</tbody>
</table>
What is CEP (Complex Event Processing)?

CEP is a technique which detects particular pattern of events from events flood, and process them.

- e.g. 1) Alert, if abnormal current values (≥ 8) are detected three times within 1 min.

- e.g. 2) Alert, if abnormal current values from different ammeters are detected five times within 1 min.
## The result: Latency

<table>
<thead>
<tr>
<th>Latency (random rule model, # of rule = 1000, attribute value range = 100)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6 # of EPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Nth=10</td>
<td>12.8</td>
<td>8.6</td>
<td>6.8</td>
<td>4.6</td>
<td>5.8</td>
<td>4.6</td>
</tr>
<tr>
<td>Proposed Nth=100</td>
<td>14.2</td>
<td>9</td>
<td>6.8</td>
<td>4</td>
<td>5.8</td>
<td>6.6</td>
</tr>
<tr>
<td>RR rule distribution</td>
<td>11.4</td>
<td>10.2</td>
<td>3.2</td>
<td>5.4</td>
<td>7.4</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latency (random rule model, # of rule = 1000, attribute value range = 10000)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6 # of EPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Nth=10</td>
<td>7.2</td>
<td>12.2</td>
<td>10.4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposed Nth=100</td>
<td>16</td>
<td>13.2</td>
<td>12</td>
<td>11.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RR rule distribution</td>
<td>17</td>
<td>12</td>
<td>11.2</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latency (car-navi model, # of rule = 996)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6 # of EPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Nth=10</td>
<td>14.8</td>
<td>10.8</td>
<td>9.4</td>
<td>7.8</td>
<td>8.2</td>
<td>7.4</td>
</tr>
<tr>
<td>Proposed Nth=100</td>
<td>15</td>
<td>10.8</td>
<td>3.8</td>
<td>4.4</td>
<td>7.6</td>
<td>5.6</td>
</tr>
<tr>
<td>RR rule distribution</td>
<td>14.6</td>
<td>10.4</td>
<td>10</td>
<td>10.8</td>
<td>5.2</td>
<td>10.6</td>
</tr>
</tbody>
</table>
The result: the number of multicast

# of multicast (random rule model, attribute value range = 100)

# of multicast (random rule model, attribute value range = 10000)

- RR
- Proposed Nth=10
- Proposed Nth=100
CUBIQ provide platform supports context-aware service by mediating among various context data and the various services.
Complex Event Processing (CEP)

Complex Event Processing consists of processing many events

Event

- Event(type1 = value1, type2 = value2. ...)
  - e.g.)
    - Event(tagID = x, readerID = y, time = t)

Rule

- Rule(Event-A, Action)
- Rule(Event-A | Event-B | ... | Event-N, Action)
- Rule(Event-A & Event-B & ... & Event-N within t, Action)
- Rule(Event-A → Event-B → ... → Event-N within t, Action)
Event Processors and Dispatchers

Event Processors (EPs) : CEP
- CEP is executed when an event arrives.
- If the event is partly-matched in some rules, the state is hold in the EP.
- If the event is matched in some rules, action in the rules executed.

Dispatchers (DISPs): Dispatch events to the EPs whose rules need the events
- DISP rules, i.e., simple event rules, are set by EP-CTL.
- Simple event processing is executed when an event arrives.
- If the event is matched in some rules, the event dispatches to the EPs which the rules shows.
  - Unicast : if the event is matched an DISP rule which specifies only one EP
  - Multicast : if the matching rule specifies more than one EPs
  - Filtered out : if there is no matching rule